Modern MSX BASIC Game Development

Build retro games in MSX BASIC using modern tools

Raul Portales

Preface

I started writing programs with an MSX when I was 10 years old. It was around 1986 and MSX BASIC was new and shiny. Back then there was no Internet and the available books were few and not very helpful. The one I used the most was just a list of all the BASIC instructions and some program listings without any explanation. It was unnecessarily hard.

Typing code from magazines, running it to see what it does, trying to follow the program, finding instructions I have never seen before, getting back to the manual to read the specifications, most of the time not very useful, and back again.

It was a terrible model for learning. The only reason I managed to understand any of it when I was a kid was pure determination, fueled by my intense fascination with computers and my natural stubbornness.

My professional career has been greatly influenced by the interest sparked by my old MSX. I am a Software Engineer -mainly working with Android- I founded a game studio in 2011 that ran for a year. I've given multiple talks internationally and even published a couple of other books; one of them about making games on Android.

And after 35 years, in 2020, for some strange reason I decided to get back to writing MSX BASIC programs. I was shocked. There were vastly more resources available now, but I still struggled to find an all in one solution with the details I wanted to know on how to write good games using MSX BASIC.

After a few months, I had a significant amount of notes, notes that I wish my past self had access to when trying to figure out how to make games; so I decided to turn them into something better, which is what this book is. This is also the book I wished I had found in 2020 when I decided that I wanted to give MSX BASIC another try. Hopefully it is useful for others that are either curious about it, or want to get back to developing for it as I did.

But why would you want to write games for a system that has been obsolete for many years? Why make it hard on yourself? And also why do it in BASIC?

There is a beauty to the sheer simplicity of a game written in MSX BASIC. You can have a game in 300 lines of code. Most modern environments require much more boilerplate code to just display a "Hello World".

Also, the concepts we will work with throughout the book (sprites, screen modes, tiles, etc) are easily transferable to other languages such as assembly. That makes BASIC a great starting language to get familiar with the ideas without the extra hardship of learning assembly.

MSX BASIC can be a very interesting entry point for making games. It is simple, yet reasonably powerful, and if you owned an MSX back in the day, it might be fun to figure out how the games were made to squeeze the hardware to its limits.

And if you are curious, there is a contest to create games for MSX run at *https://MSXdev.org*, which has been running since 2003 and there are plenty of games submitted every year. The homebrew scene of MSX is very much alive.

About MSX

If you are reading this book, you are likely to know what MSX is (and you probably even own one) so I will summarize it quickly to get to the interesting parts.

MSX is a home computer architecture built by Microsoft and ASCII in 1983. The idea was to create a standard that various manufacturers could adapt and be compatible; very similar in essence to modern PCs.

The system was very popular in Japan and Brazil, but it never caught on as much in Europe, where only Spain and the Netherlands had a significant MSX user base.

There were many manufacturers that built MSX machines, including Sony, Sanyo, Phillips, Mitsubishi, Toshiba, Panasonic, Canon, Casio, Samsung, Gradiente and Daewoo.

Dynadata was a rebrand of Daewoo with Spanish keyboard

The MSX system is quite unique as it was positioned between home computers and game consoles, having both a full keyboard and cartridge support -as well as cassette and even disk on some models- which allowed plug and play gaming, but also programming.

There are four generations of MSX. The first three, MSX (1983), MSX2 (1985), and MSX2+ were all 8-bit computers based on the Z80 microprocessor. The fourth generation, MSX TurboR, used a different 16-bit R800 microprocessor. The third and fourth generations were released only in Japan.

Versions beyond MSX-1 included more features, such as the ability to define colors, more graphical modes and even hardware scrolling. In this book we will be talking about the MSX first generation only, since what you write for it can be run on the others, but not the other way around.

The architecture of the MSX is very interesting. Besides its Z80 core microprocessor it has separate graphics and sound chips.

The graphics processor, usually referred to as VDP (Video Display Processor) has its own RAM memory which is normal nowadays, but quite unique back then. Most MSX models use the Texas Instruments TMS9918 family (same as for the ColecoVision), but there are other compatible chips like the Toshiba T6950 or the Yamaha YM2220.

Texas Instruments may not be a big name today, but they invented the hardware sprite multiplexer and even the name “sprite”. More on sprites in Chapter 4 - Game Basics.

Texas Instruments' TMS9918A was succeeded by Yamaha's V9938, which added additional bitmap modes used on MSX 2 computers.

The sound processor generally referred to as PSG (Programmable Sound Generator) is usually the General Instrument AY-3-8910 which has 3 channels and was also used by many arcade games as well as on Amstrad CPC and ZX Spectrum.

And of course, MSX computers came with MSX BASIC bundled in their ROM. MSX BASIC is a dialect of the BASIC programming language, generally designed to follow GW-Basic. It extends Microsoft's MBASIC Version 4.5, with support for graphic, music, and various MSX peripherals.

Having this language included with the computer and its amazing support for graphics (for its time) is what made many of us try it out and still want to use it today.

Book structure

The book is structured following a progressive approach in which we will cover the basics in the first chapters and will be building on top of the knowledge of previous chapters.

- In the first chapter we will talk about the development environment and the different options with their pros and cons.
- In the second chapter, we will cover the particularities of MSX BASIC as well as the different types of variables, some particularities of the language and a few recommendations on how to write programs.
- In chapter 3 we get further into MSX BASIC structure and flow as well as commands for managing input and output.
- In chapter 4, we start to build games, we will see how to define and use sprites and how to handle joysticks; and we will build a space shooter along the way.
- In chapter 5 we explore the different ways to add collision detection to our space shooter, from hardware sprite collision to hitboxes and a combination of both.
- In chapter 6, we will look into the basic drawing commands available and we will build a simple dungeon game to highlight how they can be used.
- Chapter 7 is when we get deep into the MSX graphics and dive into how to program the VDP to define tiles for more colorful and efficient graphics. This chapter is all about *VPOKE* and *VPEEK*.
- In chapter 8 we explore how we can add sound effects to our games, both using the high level command *PLAY* and also accessing the PSG directly with *SOUND*.
- Finally, in chapter 9 we will see some advanced techniques of MSX BASIC, including making it run blazing fast by compiling it with MSX-BASIC-KUN, splitting the program into many files to make the game more versatile, how to publish it, and we'll close with a brief glimpse of other more advanced ways to make games for the MSX family.

For each section, there will always be examples and small programs to run that illustrate the concepts, and the explanations will be based on the code we used. This learning by doing approach -in my experience- makes understanding the concepts and ideas much easier, since they are always applied to solving something specific.

Throughout the book there will be times when we'll see something important that you should remember. In those cases the content will be highlighted as notes which look like this:

Note	*You want to pay attention to the notes, they highlight the most important concepts and ideas.*

You can also find tips. These are used for advice, ideas or tips to make your life easier. Tips look like this:

Tip	*Tips are ideas, tricks and advice that you might want to look into when building games.*

Code will always be written in capital letters (as the original computers do) and using a monospace font to make it easier to read. A code block looks like this:

```
10 PRINT "HELLO WORLD!"
```

In addition to code blocks, when referencing BASIC commands inside the text, they will be formatted using the same font as the code blocks and also in capital letters, i.e let's talk about `GOTO`.

So, now that we have laid the groundwork, let's move into the first chapter to explore the options we have for setting up our development environment and running programs.